Formal theory of thinking

Anton Venglovskiy [0000-0001-5780-6572]

anton.venglovskiy@gmail.com

Kyiv, Ukraine, 2021

Abstract. The definition of thinking in general form is given. The constructive logic of thinking is formulated. An algorithm capable of arbitrarily complex thinking is built.

Keywords: artificial general intelligence, algorithm of mind, self-organizing algorithmic chaos.

1 Introduction

There is the so-called "the hardest logic puzzle ever". Once I solved this puzzle. In the task, you need to come up with the right question to Gods in order to find out the truth. Tackling the solution for the first time, I exhausted possible options for questions in a few days and did not come close to a solution, and in general it seemed that there could be no other questions in principle, a strange feeling: "how it happened if there is a solution, why can't I think of?" Unable to cope with the task, I postponed the decision. After a year's break, I returned to the problem and with a fresh mind noticed a way to fundamentally complicate the question for Gods, and then the solution quickly opened! Comprehending this case can give a general theoretical recommendation - you need to make the question fundamentally more complex in all senses. The questions that I could come up with in the first iteration of the solution were too simple. And the main difficulty was just how to move on to a fundamentally more complex issue. To complicate questions, I took two universal steps. First, I added new, random properties to the existing logic at all levels of abstraction, and in the second step I generalized the properties of the resulting logic, and did this in a circle many times until the generalized material reached complexity enough to express the solution. And I always observe this general scheme, in all problems for which there is no typical solution - a transition from less complexity of logic to greater complexity is required. Therefore, it seems plausible to suggest that the defining feature of thinking is the ability to create logically organized material of the necessary complexity. Subsequently, I came to a small theory with the help of which one can formally think according to the same principle by which I thought when I was solving the hardest puzzle.

2 Logic of thinking

About **thinking in general**. When considering thinking, it is impossible to abandon four fundamental characteristics: productivity, logicity, spontaneity, and complexity. If even one of these characteristics is absent, then there is no thinking. Therefore, thinking in general form is the spontaneous production of logically organized content of a certain complexity.

Logic in general. Logic implies the main, the secondary and their dynamics. There is no logic without connections and levels of abstraction. Consequently, any logically organized content can be structured as a hierarchy of interconnections, where content from higher hierarchies is more significant than content from lower ones. Transformations that preserve the content hierarchical ordering property are logical operations. Any defined set of possible sequences of logical operations is some logic. Logic will be spontaneous if the set of possible sequences of logical operations is random.

The above definition of logic is very convenient in the sense that it allows you to select a special set of objects, any operations with which are logical operations. The hierarchy of interconnections can be written literally, for example, in the following form: ((ab)c) is a rooted tree, the levels of which correspond to the levels of significance of the content, and each node of the tree contains a collection of arbitrary identifiers - which is interconnected content. For ease of writing, I skip separating characters when writing identifiers, so each letter is a separate identifier. The collection can be empty and the identifiers can be repeated. For example, there may be trees like this: (a), ((aa)(aa)), ((a)(b)), (ab(cd(efg(a)))), (ab(ab)(ab)(aabb)), (a(b(cccc(dd)(ee)))) and the like. It is convenient to operate with such trees as strings, so from now on I will speak simply - strings. Isomorphism's of strings are identical, for example, ((a(b))c) ≡ (c((b)a)). On the set **S** of all possible strings of the specified type, any operations of the type **S** → **S** will be logical operations, so any sequence of string transformations will correspond to some kind of logic. This logical universality of content is suitable for the logic of thinking in general.

The logic of thinking. The logic of thinking assumes arbitrary complexity. There is no thinking if there is no possibility of transition from less complexity of logic to more complexity. Arbitrarily complex logic can be implemented using two universal operations. As I noted in the introduction, in the process of solving the "hardest puzzle", I was doing two types of iterations: adding random properties in the logic of questions and generalizing. Consequently, for the logic of thinking, an operation of generalization is necessary - an Abstraction that deduces the main and discards the secondary, so that theories of things can be built. And also a operation for adding random properties to different levels of the logic hierarchy is necessary – an Deduction, which adds new principles to logic, so that complex theories of things can be built. Thus, two corresponding operations - Abstraction and Deduction on the set **S** define the logic of thinking.

Operation of Abstraction. Acts on the principle of detecting a common property for a group of objects. On the set **S**, this principle can be expressed literally: ((ab)(ac)) ⇒ (a(bc)). The general letter "a" can be put out of the parentheses, thus raising its level in the hierarchy of abstractions, we can say that the letter "a" has become a generic feature for everything lower in the hierarchy, and the remnants are merged in common parentheses because they are interconnected through the generic sign "a". After the action of the operation, the construction of the string is "simplified", the loss of insignificant information (compression) occurs, which is also characteristic of natural generalization procedures. The rule of Abstraction can be generalized. The rule is valid for any substrings, any given string. If any group of substrings is at the same level of the hierarchy, for example, (...()()...), (...()()()...), (...()()()()...), ..., (...(...()()...)...), (...(...()()()...)...), ..., and if any combination of substrings contains the same content, for example, like this (...(ab)(aabb)...), or this (...(ab)(aabc)(aabbcc)...), then any piece of matching content can be put out of the parentheses, for example like this ((ab)(aabc)(aabbcc)) ⇒ (ab(ac)(aabbcc)), or like this ((ab)(aabc)(aabbcc)) ⇒ (a(babcabbcc)), or this ((ab)(aabc)(aabbcc)) ⇒ ((ab)abc(aabc)), ((ab)(aabc)(aabbcc)) ⇒ (ab(aabccc)) and there are still many options left. The content that is putted out from the parentheses is underlined. As you can see, after putting out the general content, the remnants in the parentheses must be merged. You can put out from the parentheses not only identifiers, but also substrings, for example, (((a(b))(c(b))) ⇒ ((b)(ac)), and if there are empty parentheses, they must be discarded, for example, (((a(b))(a(b))) ⇒ (a(b)()) ≡ (a(b)). Since a string can be simplified in many alternative and mutually exclusive ways, the Abstraction rule must be performed sequentially, that is, in one step, you can make only one simplification for any one group of parentheses and one selected piece of content. Using the Abstraction rule, for any given string, you can sequentially construct the set of all possible simplifications in all possible alternative ways, using recursion. For example, consider the string ((a)(bc(a))(bc(a))), and write down all possible simplifications: on the first iteration its be ((a)(bc(a))(bc(a))) ⇒ (b(a)(cc(a)(a))), ((a)(bc(a))(bc(a))) ⇒ (c(a)(bb(a)(a))), ((a)(bc(a))(bc(a))) ⇒ (bc(a)((a)(a))), ((a)(bc(a))(bc(a))) ⇒ ((a)(a)(bbcc)), ((a)(bc(a))(bc(a))) ⇒ ((a)bc(a)), now apply the Abstraction rule to the results obtained at the first iteration, where possible (b(a)(cc(a)(a))) ⇒ (b(a)(cca)), (c(a)(bb(a)(a))) ⇒ (c(a)(bba)), (bc(a)((a)(a))) ⇒ (bc(a)(a)), ((a)(a)(bbcc)) ⇒ (a(bbcc)), ((a)bc(a)) ⇒ (abc), again apply the Abstraction to the results of the second iteration (b(a)(cca)) ⇒ (ab(cc)), (c(a)(bba)) ⇒ (ac(bb)), (bc(a)(a)) ⇒ (abc), that's all, there are no more options, the procedure recursively converged. Thus, with the help of recursion, we have decomposed the source, complex string into many simple and prime strings. Recursive decomposition of the source string into a set of prime ones using the Abstraction rule, I will call the Abstraction operator A: **S**→2^**S**, to summarize, we write, A[((a)(bc(a))(bc(a)))] = {(b(a)(cc(a)(a))), (c(a)(bb(a)(a))), (bc(a)((a)(a))), ((a)(a)(bbcc)), ((a)bc(a)), (b(a)(cca)), (c(a)(bba)), (bc(a)(a)), a(bbcc)), (abc), (ab(cc)), (ac(bb)), (abc)}, the specified multiset of Abstraction results contains all possible generalizations of the source string.

Deduction operation. Adds new constructive properties to the source hierarchy, extends, and always complicates the interconnections hierarchy. On set **S**, there is a

simple and natural way to define Deduction. First, note that strings can be written in a more compact form if repeating elements are written using a multiplier prefix, for example, (aa) ≡ (2a), ((a)(a)) ≡ (2(a)), ((aa)(aa)) ≡ (2(2a)), (aa(bb(ccc)(ccc))(bb(ccc)(ccc))) ≡ (2a2(2b2(3c))), in this notation any string is unambiguously unpacked from left to right. The Deduction rule duplicates all elements of the source string, any given number of times, and is general case defined as follows: (a) ⇒ {(2(2a)), (3(3a)), (4(4a)), ...}, that is, from any source string, you can get an infinite number of Deductions that will differ in the number of duplicates of the source elements. For practical purposes, in order to write down the algorithm, the number of duplicates must be fixed, so I define the Deduction operator as D: **S**→**S**, D[(a)] = (2(2a)). More examples: D[(aa)] = (2(4a)), D[((a)(b))] = (2(2(2a)2(2b))), D[(a(b(c)))] = (2(2a2(2b2(2c)))) and the like. As you can see, even if the source string is prime relative to the Abstraction rule, then its Deduction will always be complex, that is, it will have a non-empty decomposition into prime strings, for example, A[D[(a)]] = A[(2(2a))] ≡ A[((aa)(aa))] = {(aa), (a(aa))}, from this example it is obvious that the source string (a) after Deduction acquired fundamentally new constructive properties that Abstraction detects.

Complexity. Complex things have more generalizations than simple things. The logic of an object is more complex the more different generalizations can be established for a given object. The Abstraction operator naturally defines the complexity on the set **S**. The more prime strings in the decomposition of source string using the Abstraction operator, the more complex the source string is. I define complexity as C[s] = |A[s]|, where |A[s]| is the number of elements in the multiset of Abstraction results. The operator C can be used to express the main property of Deduction, C[s] < C[D[s]], which is obviously by construction. The complexity of all prime strings is the same and does not depend on their size and hierarchy, for example, C[(a)] = C[(a(b)(c))] = C[(a(b(c))(c(a)))]. However, for prime strings, you can define their potential complexity through Deduction, C[D[(a)]] < C[D[(a(b)(c))]] < C[D[(a(b(c))(c(a)))]], in which case the potential complexity will depend on the size and structure of the source prime strings.

Thinking process. On the set **S**, Abstraction and Deduction procedures are given, as well as complexity is defined. Thus, the logic of thinking is set. Since the specific level of complexity of human thinking is unknown, it is logical to assume that a computational process satisfies the criteria for real thinking if it can create logically organized content of arbitrary (not limited from above) complexity, and under the assumption of infinite computations, the complexity of the behavior of such a process should be infinite. To formally write such a process, an additional string concatenation operator is required. For syntax uniformity, the concatenation operator will be written as parentheses ({...}), the concatenation operator can be applied to any set of strings and result in one string, for example, (A[D[(a)]]) = (A[((aa)(aa))]) = ((aa)(a(aa))). As you can see, many of the two Abstraction results have been concatenated into one string. So, the thinking process in general form: t_n = (A[D[t_{n-1}]]); $t_0 \in$ **S**. This recursive function produces logical, unique and arbitrarily complex content in any quantity from any nonempty starting value.

Spontaneity. Thinking is spontaneous, no one can predict their next thought. In thinking, temporal trends can be identified, so you can predict the direction of thoughts to some extent, for example, you can roughly guess what you will think tomorrow, but such predictions are mostly wrong since the origin of trends is also accidental. Collective forms of thinking, such as the development of science or the history of society, are also clearly random. No one can reliably predict history or how this or that scientific theory will develop. Spontaneity is the guarantee of any fundamental novelty and this property is not reducible to the effect of a set of external factors, but it is inherent in thinking, which is obviously observed in closed subjective thinking. Thereby the computational process of thinking must be algorithmically random. In a strict sense, I believe that the set of all possible strings that the function t_n enumerates at the abstraction stage is undecidable, and therefore the function t_n is algorithmically random, but I have no proof.

At this stage of reasoning, the function t_n fully satisfies the above definition of thinking. This function produces spontaneous (algorithmically random), logically organized content of any given complexity. Moreover, the complexity is defined explicitly and constructively, and thanks to the Deduction operator, it increases iteratively, it can be calculated and evaluated. The logical organization of content is guaranteed by the syntax and the Abstraction operator, which iteratively generalizes the logic of the produced content. Thus, the function t_n represents an algorithmic form of thinking in its pure, ideal form. Calculating this function is thinking.

Adjustment of generalizing ability and complexity of thinking. As you can see from the construction, the complexity of the content produced by the t_n function increases exponentially, this is impractical. However, there is a natural way to regulate the generalization and complexity of content over a wide range. Any fixed level of complexity and abstractness of content can be maintained. Due to the specific action of Abstraction, highly organized content floats to the upper levels of the hierarchy, while at the lower levels of the hierarchy, more chaotic content remains, that is, the content is ordered by the level of significance, generality and logical organization. Therefore, it is possible to discard the lower, insignificant levels of the hierarchy, that is, you can perform additional generalization of content using a cruder method. For example, from the string (abc(de(fk(gh)(cn(rt))))) you can extract the most significant part by cutting off deep nesting levels, like this (abc(de(fk(gh)(cn(rt))))) ⇒ (abc(de(fk))), in showed case a relatively insignificant fragment (gh)(cn(rt)) was removed. The procedure for removing insignificant content can be built into the t_n function after the Abstraction stage, like this t_n = (Truncate[A[D[t_{n-1}]], d]), where d is the depth of the removed content. In addition to deletion of deep nesting, at the Abstraction stage it is also possible to selectively leave the Abstraction results, for example, it may make sense to leave only prime strings since they are the most generalized, and filter out the rest. In general, it is possible to filter the results of Abstraction according to their statistical significance for each specific practical problem.

Practical use. In order for the described formal thinking process to acquire a meaningful character, it is necessary to feed it a meaningful external signal, encoded with strings, according to the following scheme: t_n = (A[D[((t_{n-1})(InputSignal$_n$))]]), if a cer-

tain meaning is encoded in the input signal, then the behavior of the t_n process will also have a certain meaning. The t_n process thinks abstractly in its own internal language, in order to understand the content of such thinking, it is necessary to build a model of a common language with the help of a feedback loop.

Intelligence. I consider intelligence as an epiphenomenon. That is, intelligence is a dependent, partial and optional form of the general process of thinking. This form of thinking can arise in response to a specifically intellectual interaction with the general process of thinking. All specifically intellectual phenomena like goal-setting, pattern recognition, predictive ability, adaptability, and problem-solving ability are the result of a more universal logic, which I call the logic of thinking. This logic does not depend on intelligence and its manifestations. Everything secondary is deliberately discarded and the very phenomenon of intelligence is considered secondary in relation to a much more abstract process of self-organizing algorithmic chaos t_n, which infinitely evolves through generalizations. For intelligence to arise in the t_n process, the process must be viewed as a thinking black box, a kind of "alien intelligence", and it is necessary to implement meaningful, intelligent interaction using modern machine learning methods.

3 Conclusion

The main proposal is to free thinking from any kind of a priori phenomenology, from things, actions and meanings. Can is seen that thinking in general form does not need these concepts. Things and the logic of things are secondary products of thinking. For an accurate definition of thinking, the idea is important that the mind is a very high-level process, it does not depend on the meaning of what it creates, and therefore does not need the concept of meaning in its own definition.

From the point of view that thinking does not depend on meaning, one can proceed to the formal definition of thinking through complexity and then to its algorithmization - to a universally thinking algorithm. To do this, it should be noted that thinking has a universal product - this is complexity. All human theories of things are complex. This means that we can assume that the mind achieves its goals at the level of complexity of things, and the meaning of things is a low-level, by-product, and therefore thinking does not depend on meaning. The main question to which thinking answers (without meaning) is how to make other fundamentally new, more complex things from a given, simple thing. Thus, all transformations of thought are transformations of the constructive complexity of things. However, for an outside observer, the phenomenology of thinking can look infinitely vast, because transformations at the level of complexity create a diverse, new logic of things.

In turn, the transformations of things at the level of complexity can be algorithmized in a general and explicit form. And the resulting algorithm, therefore, will be an accurate, constructive definition of mind in a general form. The main task of the algorithm is to create a new logic of things, of any required complexity. For this, the concept of logic is generalized and things are divided into logical and illogical. The algorithm

creates only logical things and therefore they can be investigated and comprehended in concepts corresponding to the logical complexity of things.

The algorithm can be interacted with, and the algorithm is able to independently comprehend the nature of interaction at various levels of abstraction in the subjective system of concepts, which, along with the ability to evolve, is a prerequisite for high-level communication and intelligence. Intelligence is seen as a form of thinking that arises in response to meaningful interaction. In other words, any meaningful interaction, as a result of generalization, structures the subjective logic of the algorithm as the logic of intelligence. Thus, the initial assumption that thinking in its pure form does not need meaning does not exclude the possibility of meaning and intelligence and even turns out to be a prerequisite for their emergence and existence.